Parallelized and Vectorized Image Enhancing Algorithm for Scanning or Printing

Algorithm Engineering 2025 Project Paper

Angelo Wieden
Friedrich Schiller University Jena
Germany
angelo.wieden@uni-jena.de

Abstract

This paper deals with the development of a fast, platform-independent program for image enhancement based on local pixel data. The output produces a clean black and white version of the input picture, which results in the appearance of a scanned image. This research paper aims to answer the following question. How can such an algorithm be efficiently implemented in C++, and which computational parameters could be manually tweaked on a picture-by-picture basis, for best results? The implementation of the image enhancement algorithm was done in C++, using OpenMP to parallelize and vectorize specific sections. Bataineh's binarization algorithm served as a foundation, which was built upon. CMake was used to build and package the software. Testing was done with the help of Catch2. The standalone stb library helped in reading and writing certain image file types. The parallelized version achieves speedups of up to 12x over the serial version. These findings demonstrate the importance of writing efficient code for modern multicore CPUs, in the area of image processing.

Keywords

image enhancing, algorithm engineering, parallelization, vectorization

1 Introduction

1.1 Background

Image enhancement is a broad field of digital image processing that covers a variety of tasks. Although it is mainly used to make images more readable for humans, image enhancement also finds various applications in other areas such as preprocessing for OCR or medical imaging. Traditionally, image enhancement is done serially, wasting a large amount of computational power of modern multicore CPUs. An already existing image enhancing algorithm was modified within the scope of the 2025 Algorithm Engineering Project, using full advantage of SIMD vectorization and parallelization through OpenMP for large performance gains. This particular algorithm uses binarization as a form of enhancement, which is the process of turning pixels either fully white or black depending on different factors.

1.2 Related Work

The algorithm that was implemented and built upon was designed on the basis of the adaptive threshold calculation used for pixel binarization by Bataineh [1]. A different binarization approach is taken by Bradley [2], which will be compared to ours in section 3.

1.3 Our Contributions

Aside from the implementation of the algorithm itself in C++, there were other means necessary like input parsing, output handling, implementing various argument flags for manual parameter tweaking and benchmarking, as well as a lot of fine-tuning, writing unit tests with Catch2 and making the program as portable as possible with the help of CMake.

1.4 Outline

In the following section of this paper, an explanation on how the algorithm decides which pixels should be assigned which color will be provided, as well as information on how input files are parsed, and output is handled. Furthermore, the modifications that were done to the algorithm by Bataineh will be explained. [1].

Section 3 shows the before-and-after experiments of enhanced documents by the proposed program. Also, there will be a comparison of our implementation and the original algorithm. At last, a performance benchmark will show the resulting speedup of the serialized and optimized versions of the algorithm on two different modern systems.

In section 4, the reader will have a look at a discussion of findings stemming from this project, and an analysis about future directions.

2 The Algorithm

This section explains the inner workings of the program.

2.1 Processing of Input- and Output Files

Upon receiving an input file, parsing happens dependent on the input file's type. The stb library handles both .jpg and .png files, and converts them to an array of color values, where each pixel is assigned 3 - red, green and blue - values ranging from 0 to 255. The same formatting is done for .ppm files through a simple loop, as the file structure of .ppm allows for easy parsing.

As a last preprocessing step before the binarization algorithm can start, the formatted pixel color values have to be mapped to a singular grayscale value by taking the average of the red, green and blue channels on a per-pixel basis.

To generate output files, the stb library is used for .jpg and .png files, while .ppm file writing was manually implemented.

2.2 Binarization Algorithm

The C++ algorithm works by adaptively assigning each pixel a threshold of a color value, based on multiple factors. These factors are heavily influenced by the target's neighboring pixels within a window around the target. If the original image's grayscale value is less than this threshold, the enhanced image's target pixel will be black. Otherwise, the pixel will turn white.

To calculate the threshold value of a pixel, a series of calculations need to be done:

(1) **Mean Color Value of the Image:**

$$\mu = \frac{1}{N}\sum_{i=1}^{N} I_i \tag{1}$$

where μ is the mean color value, N is the total number of pixels, and I_i represents the grayscale intensity of the i-th pixel.

(2) **Each Pixel's Deviation from the Mean:**

$$D_i = |I_i - \mu| \tag{2}$$

where D_i is the absolute deviation of pixel i from the mean.

(3) **Standard Deviation of the Image:**

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(I_i - \mu)^2} \tag{3}$$

where σ is the standard deviation.

(4) **Minimum and Maximum Deviation Values:**

$$D_{\text{min}} = \min_i D_i, \quad D_{\text{max}} = \max_i D_i \tag{4}$$

where D_{min} and D_{max} represent the minimum and maximum deviations, respectively. It should be noted that the original algorithm by Bataineh [1] used the minimum and maximum values of all windows. Testing revealed that this approach provides similar results with reduced computational load.

(5) **Window Borders and Number of Pixels within the Window:**

$$y_{\text{start}} = \max(0, y_i - \frac{W_{\text{size}}}{2}) \tag{5}$$

$$x_{\text{start}} = \max(0, x_i - \frac{W_{\text{size}}}{2}) \tag{6}$$

$$y_{\text{end}} = \min(y_i + \frac{W_{\text{size}}}{2}, h - 1) \tag{7}$$

$$x_{\text{end}} = \min(x_i + \frac{W_{\text{size}}}{2}, w - 1) \tag{8}$$

$$N_{\text{window}} = (x_{\text{end}} - x_{\text{start}} + 1) \times (y_{\text{end}} - y_{\text{start}} + 1) \tag{9}$$

where W_{size} is the window size, h and w are the image height and width, and N_{window} is the total number of pixels in the window centered at pixel i.

(6) **Mean Value of Pixels in a Window:**

$$\mu_w = \frac{1}{N_{\text{window}}}\sum_{j\in W} I_j \tag{10}$$

where μ_w is the mean grayscale value within the window centered at pixel i, and W is the set of pixels within the window.

(7) **Standard Deviation within a Window:**

$$\sigma_w = \sqrt{\frac{1}{N_{\text{window}}}\sum_{j\in W}(D_j - \mu_w)^2} \tag{11}$$

where σ_w is the standard deviation within the window centered at pixel i.

(8) **Adaptive Deviation for Each Window:**

$$A_i = \frac{\sigma_w - D_{\text{min}}}{D_{\text{max}} - D_{\text{min}}} \tag{12}$$

where A_i is the adaptive deviation for the window centered at pixel i.

(9) **Finally, the threshold can be calculated as:**

$$T_i = \left(\mu_w - \frac{\mu_w^2 - \sigma_w}{(\mu + \sigma_w)\cdot(A_i + \sigma_w)}\right) \times M \tag{13}$$

This calculation features a noise multiplier M, which is an addition to the original algorithm.

Upon observing that some pictures needed additional preprocessing to reduce general image noise, it was found out that multiplying the threshold values by a constant between 0 and 1 will reduce more noise, the lower the constant is. The reduced noise will also result in thinner lines though, which is why this should be set to the highest value possible that results in minimal image noise. A higher window size seems to make thin lines thicker, but also has the biggest negative impact on runtime.

These steps have been successfully parallelized and vectorized through OpenMP. All necessary variables have been aligned to 64-bit registers for optimal performance. After calculating the threshold, the final image can be constructed based on the binarization logic above. By default, the window size is set to 20 and the noise multiplier is set to 0.9, which should work well for most images.

2.3 Program Flags

Multiple flags are included to manually tinker with parameters or gain more insight on the program. These are:

- `-h` for information about program usage and flags.
- `-v` activates verbose mode, printing information during the algorithm process.
- `-b` gathers the elapsed time (excluding the input processing and output writing phases)
- `-s` activates the serial mode, allowing only one thread.
- `-o="outputDir/outputName.filetype"` to assign the output folder and filename.
- `-w=INT` specifies the window size.
- `-n=INT` sets the noise multiplier in % (0 to 100) -> (0 to 1).

If no flags are set, the program will use a standard configuration.

2.4 Build and Compilation

For build automization, CMake was chosen for cross-platform compatibility. Detailed instructions on how to build the program are in the projects readme file.

It is highly advised to use GNU or Clang as the compiler, as MSVC does not fully support newer OpenMP functions as of now. Full MSVC support was still added in CMake and the program itself for a flawless integration once MSVC is fully compatible with OpenMP.

The program is compiled with the following flags:

- `-fopenmp -O2 -march=native -ffast-math -std=c++17` for GNU / Clang
- `/openmp /O2 /arch:AVX /fp:fast /std:c++17` for MSVC

2.5 Program Usage

To enhance an image, execute the program and pass the path to the image you would like to enhance, as well as any arguments:
`./main path_to_image <args>`
If the output image has too much noise or the text lines appear too thin, adjust the program flags by reducing the noise multiplier or increasing the window size until the desired look is achieved.

2.6 Unit Tests

The compiled `catch_tests_image_enhancer.cpp` file can be executed to perform all unit tests, which include more than 200 assert operations on an example 5x5 pixel testing image with a window size of 3. These tests include edge cases as well as pre-calculated expected values for all necessary functions.

3 Experiments

3.1 Image Enhancement Examples

To show the potential of this algorithm, two comparisons against other image enhancing algorithms are provided. Figure 1 compares our implementation with the original thresholding algorithm by Bataineh [1]. In figure 2, our implementation gets compared to a different thresholding algorithm by Bradley [2], which calculates the thresholds using integral images.

Lastly, figure 3 shows that the algorithm is fully capable of enhancing documents in a real-world scenario, with default parameters.

3.2 Runtime Comparison

Tables 1 and 2 show the performance difference of the serial and parallel algorithm based on images 2 and 3, as they have different resolutions. Multiple window sizes were chosen to benchmark, as an increase in window size leads to an exponential increase in computational work. Luckily, in everyday scenarios, e.g. enhancing a picture of a printed email, window sizes can be chosen as small as 10-15. To measure the serial speed the program was executed with the `-s` flag, ensuring that only the computational power of one thread is being used.

All measurements were taken with the `-b` benchmarking flag under the same conditions, and the laptop was being charged at all times, making sure that no throttling was present. In both tables 1 and 2, we can see a runtime speedup of up to `11x` on the desktop computer, while the laptop gets a maximum speedup of `8x` compared to the serial version.

On both machines, we can observe a trend that the lowest window size benchmarked has the least speedup of `8x` for desktop and `4x` for laptop. These diminishing returns are likely caused by the parallelization overhead slowly outweighing the speedup of parallelized work for smaller workloads.

Table 1: Comparison of the running times and speedup of the image enhancement algorithm based on two pictures and three different window sizes per picture. Measurements were taken on a desktop pc running Windows 10 Version 22H2 running on an AMD Ryzen 7 5800X (8 cores, 16 threads) CPU. Speedup is measured as the ratio of the execution time of the serialized version to that of the parallelized version.

Image	Resolution	Window Size	Serial (s)	Parallel (s)	Speedup
Image_2	1210×904	20	0.455	0.056	**8.07**
		50	2.703	0.249	**10.84**
		100	10.916	1.031	**10.59**
Image_3	1536×2048	20	1.328	0.152	**8.75**
		50	7.814	0.71	**11.01**
		100	31.977	3.038	**10.53**

Table 2: Comparison of the running times and speedup of the image enhancement algorithm based on two pictures and three different window sizes per picture. Measurements were taken on a laptop running Windows 11 Version 24H2 running on an Intel Core i5-12500H (12 cores, 16 threads) CPU. Speedup is measured as the ratio of the execution time of the serialized version to that of the parallelized version.

Image	Resolution	Window Size	Serial (s)	Parallel (s)	Speedup
Image_2	1210×904	20	0.349	0.087	**3.99**
		50	1.969	0.249	**7.9**
		100	7.5	1.363	**5.5**
Image_3	1536×2048	20	1.044	0.201	**5.19**
		50	5.969	0.984	**6.07**
		100	22.801	3.635	**6.27**

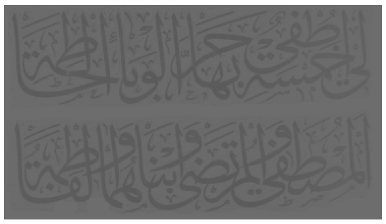
4 Conclusions and Future Work

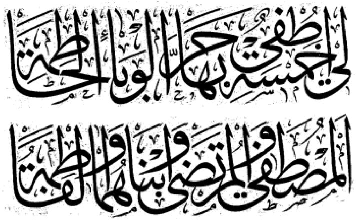
Based on the runtime benchmark results, the time needed to enhance even a high resolution picture seems reasonable. This is further amplified by the fact that in real-world scenarios, the window size can often be chosen as low as 10-15 without implications. Therefore, the overall image enhancement project can be seen as a success.

A considerable amount of time has been spent researching about ways to measure the noise level of an image, with the end goal of finding a way to implement an automatic estimation for the best noise multiplier and window size parameters per picture. While there are multiple methods to compute an estimation of such noise level, a solution has yet not yet been found to either:

(1) approximate good parameters or
(2) automatically try out different parameters until a good noise level is reached.

While the second option seems easy, it is easier said than done as all pictures have different preferred noise levels for optimal enhancement. If further work were to be done on this project, the main


(a) Image 1


(b) Output 1, Bataineh's implementation [1]

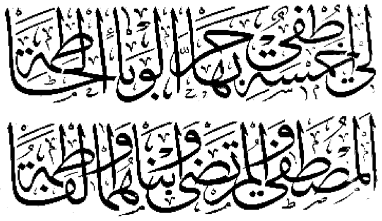
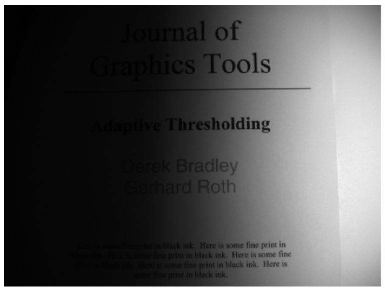

(c) Output 1, our implementation (n=96 w=75)

Figure 1: A comparison to Bataineh's implementation [1]

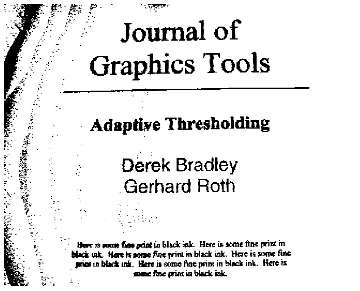

Journal of
Graphics Tools

Adaptive Thresholding

Derek Bradley
Gerhard Roth

Here is some fine print in black ink. Here is some fine print in black ink. Here is some fine print in black ink. Here is some fine print in black ink.

(a) Image 2

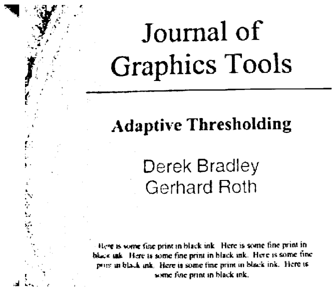

Journal of
Graphics Tools

Adaptive Thresholding

Derek Bradley
Gerhard Roth

Here is some fine print in black ink. Here is some fine print in black ink. Here is some fine print in black ink. Here is some fine print in black ink.

(b) Output 2, Bradley's implementation [2]


Journal of
Graphics Tools

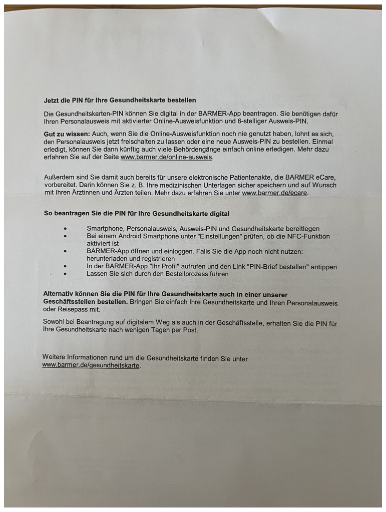
Adaptive Thresholding

Derek Bradley
Gerhard Roth

Here is some fine print in black ink. Here is some fine print in black ink. Here is some fine print in black ink. Here is some fine print in black ink.

(c) Output 2, our implementation (n=80 w=35)

Figure 2: A comparison to Bradley's implementation [1]


Jetzt die PIN für Ihre Gesundheitskarte bestellen

Die Gesundheitskarten-PIN können Sie digital in der BARMER-App beantragen. Sie benötigen dafür Ihren Personalausweis mit aktivierter Online-Ausweisfunktion und 6-stelliger Ausweis-PIN.

Gut zu wissen: Auch, wenn Sie die Online-Ausweisfunktion noch nie genutzt haben, lohnt es sich, den Personalausweis jetzt freischalten zu lassen oder eine neue Ausweis-PIN zu bestellen. Einmal erledigt, können Sie dann künftig auch viele Behördengänge einfach online erledigen. Mehr dazu erfahren Sie auf der Seite www.barmer.de/online-ausweis.

Außerdem sind Sie damit auch bereits für unsere elektronische Patientenakte, die BARMER eCare, vorbereitet. Darin können Sie z. B. Ihre medizinischen Unterlagen sicher speichern und auf Wunsch mit Ihren Ärztinnen und Ärzten teilen. Mehr dazu erfahren Sie unter www.barmer.de/ecare.

So beantragen Sie die PIN für Ihre Gesundheitskarte digital

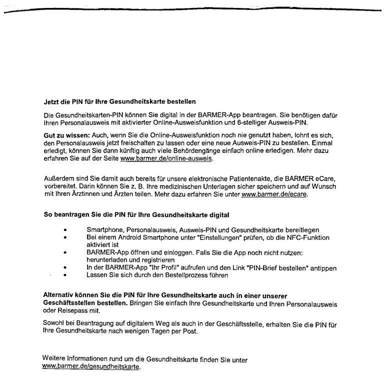
- Smartphone, Personalausweis, Ausweis-PIN und Gesundheitskarte bereitlegen
- Bei einem Android Smartphone unter "Einstellungen" prüfen, ob die NFC-Funktion aktiviert ist
- BARMER-App öffnen und einloggen. Falls Sie die App noch nicht nutzen: herunterladen und registrieren
- In der BARMER-App "Ihr Profil" aufrufen und den Link "PIN-Brief bestellen" antippen
- Lassen Sie sich durch den Bestellprozess führen

Alternativ können Sie die PIN für Ihre Gesundheitskarte auch in einer unserer Geschäftsstellen bestellen. Bringen Sie einfach Ihre Gesundheitskarte und Ihren Personalausweis oder Reisepass mit.

Sowohl bei Beantragung auf digitalem Weg als auch in der Geschäftsstelle, erhalten Sie die PIN für Ihre Gesundheitskarte nach wenigen Tagen per Post.

Weitere Informationen rund um die Gesundheitskarte finden Sie unter www.barmer.de/gesundheitskarte.

(a) Image 3


Jetzt die PIN für Ihre Gesundheitskarte bestellen

Die Gesundheitskarten-PIN können Sie digital in der BARMER-App beantragen. Sie benötigen dafür Ihren Personalausweis mit aktivierter Online-Ausweisfunktion und 6-stelliger Ausweis-PIN.

Gut zu wissen: Auch, wenn Sie die Online-Ausweisfunktion noch nie genutzt haben, lohnt es sich, den Personalausweis jetzt freischalten zu lassen oder eine neue Ausweis-PIN zu bestellen. Einmal erledigt, können Sie dann künftig auch viele Behördengänge einfach online erledigen. Mehr dazu erfahren Sie auf der Seite www.barmer.de/online-ausweis.

Außerdem sind Sie damit auch bereits für unsere elektronische Patientenakte, die BARMER eCare, vorbereitet. Darin können Sie z. B. Ihre medizinischen Unterlagen sicher speichern und auf Wunsch mit Ihren Ärztinnen und Ärzten teilen. Mehr dazu erfahren Sie unter www.barmer.de/ecare.

So beantragen Sie die PIN für Ihre Gesundheitskarte digital

- Smartphone, Personalausweis, Ausweis-PIN und Gesundheitskarte bereitlegen
- Bei einem Android Smartphone unter "Einstellungen" prüfen, ob die NFC-Funktion aktiviert ist
- BARMER-App öffnen und einloggen. Falls Sie die App noch nicht nutzen: herunterladen und registrieren
- In der BARMER-App "Ihr Profil" aufrufen und den Link "PIN-Brief bestellen" antippen
- Lassen Sie sich durch den Bestellprozess führen

Alternativ können Sie die PIN für Ihre Gesundheitskarte auch in einer unserer Geschäftsstellen bestellen. Bringen Sie einfach Ihre Gesundheitskarte und Ihren Personalausweis oder Reisepass mit.

Sowohl bei Beantragung auf digitalem Weg als auch in der Geschäftsstelle, erhalten Sie die PIN für Ihre Gesundheitskarte nach wenigen Tagen per Post.

Weitere Informationen rund um die Gesundheitskarte finden Sie unter www.barmer.de/gesundheitskarte.

(b) Output 3, our implementation (default parameters)

Figure 3: A real-world example.

goal would be to find correlations between image parameters like mean color, standard deviation, min deviation and max deviation in combination with the preferred noise levels for maximum enhancement. A further approach would be to train a machine learning model to estimate noise levels at which an image would look the best.

References

[1] Bilal Bataineh, Siti N. H. S. Abdullah, K. Omar, and M. Faidzul. 2011. Adaptive Thresholding Methods for Document Image Binarization. In *Mexican Conference*

on Pattern Recognition (MCPR 2011) (Lecture Notes in Computer Science, Vol. 6718), J.-F. Martínez-Trinidad et al. (Ed.). Springer-Verlag Berlin Heidelberg, 230–239. https://link.springer.com/content/pdf/10.1007/978-3-642-21587-2_25.pdf

[2] Derek Bradley and Gerhard Roth. 2007. Adaptive Thresholding Using the Integral Image. In *Proceedings of the International Conference on Computer Vision Theory and Applications (VISAPP)*. INSTICC, Canada. https://people.scs.carleton.ca/~roth/iit-publications-iti/docs/gerh-50002.pdf